Graph Edge Bundling by Medial Axes

Ersoy, Ozan; Telea, Alexsandru

Published in:
Proceedings ASCI/IPA/SIKS

Publication date:
2011

Graph Edge Bundling by Medial Axes

Ozan Ersoy
Institute Johan Bernoulli, University of Groningen
o.ersoy@rug.nl

Alexandru Telea
Institute Johan Bernoulli, University of Groningen
a.c.telea@rug.nl

Abstract

We present a new method for bundling edges of general graphs, based on 2D medial axes of edge sets which are similar in terms of position. We combine edge clustering, distance fields, and 2D medial axes to progressively bundle general graphs by attracting edges towards the centerlines of level sets of their distance fields. Our method allows for an efficient GPU implementation. We illustrate our method on several large real-world graphs.

1 Introduction

Graphs are among the most important data structures in information visualization. Classical visualization metaphors for graphs include node-link diagrams [16], matrix plots [29], and graph splatting [30]. For specific types of graphs, such as hierarchies (trees), additional methods exist such as treemaps.

As the size of a graph increases, node-link visualizations are challenged by *clutter*, *i.e.* unorganized groups of nodes and edges onto small screen areas. To reduce clutter, and also help the simplified depiction of large graphs with an emphasis on structure, several methods have emerged [11], such as edge bundling. Bundling starts with node positions, either given or computed by a layout algorithm. Edges found to be close in terms of graph structure, endpoints position, data attributes, or combinations thereof, are drawn as tightly bundled curves. This trades clutter for overdraw and produces images which better emphasize the graph structure. Bundles can be rendered using various effects such as blending or shading [15, 20, 27].

In this paper, we present a novel approach for bundling general graphs. We combine edge clustering, distance fields, and 2D medial axes to construct bundled layouts by iteratively attracting edges towards the centerlines of level sets of their distance fields. Besides clustering, we work image-based, with an efficient implementation in graphics hardware. Our method creates strongly branching (organic-like) bundles which always have a tree structure. This type of look emphasizes how several edges 'join' together into, or split from, main structures.

In Section 2, we review related work on edge bundling. Section 3 presents our new algorithm. Section 4 details implementation. Section 5 presents examples on large real-world graphs. Section 6 discusses our method. Section 7 concludes the paper.

2 Related work

Reducing clutter in graph visualization is organized as follows.

Graph simplification techniques reduce clutter by simplifying graphs prior to layout *e.g.* by grouping strongly connected subgraphs into so-called metanodes [1, 2]. This reuses existing node-link layouts out of the box, but can be sensitive to simplification parameters, which further depend on the type of graph being processed. Also, the simplification events yield a set of discrete graphs rather than a smooth exploration scale [20]. Finally, simplification changes node positions (collapse to metanodes), which is unwanted when positions encode information.

Edge bundling techniques trade clutter for overdraw, by routing geometrically and semantically related edges along similar paths. This helps visually finding groups of nodes related to each other by groups of edges (the bundles). Dickerson *et al.* merge edges by reducing non-planar graphs to planar ones [9]. Holten pioneered edge bundling for compound (hierarchy-and-association) graphs by routing edges along the hierarchy layout using B-splines [14]. Gansner and Koren use area optimization metrics [13] to bundle edges in a circular layout similar to [14]. Dwyer *et al.* use curved edges in force-directed layouts to minimize crossings, which creates bundles [10]. Force-directed edge bundling (FDEB) creates bundles by attracting control points on close edges [15]. FDEB can be optimized using multilevel clustering techniques [12]. For directed graphs, flow maps use a binary clustering of nodes to route edges [21]. Several methods use control meshes to route curved edges, *e.g.* [22, 31]; a Delaunay-based extension called geometric-based edge bundling (GBEB) [7]; and 'winding roads' (WR) which use Voronoi diagrams for 2D [20] and 3D [19] layouts.

3 Algorithm

The medial axis, or skeleton, of a 2D shape is a curve locally centered with respect to the shape's boundary [6]. Skeleton branches capture well the topology of elongated shapes [18, 24]. Hence, if we could create such shapes from sets of graph edges, their skeletons could be suitable locations for bundling. To this end, our proposed skeleton-based edge bundling method is as follows:

1. *cluster* edges into groups, or clusters, C_i which have strong geometrical and optionally attribute-based similarity;
2. for each cluster C, compute a thin shape Ω surrounding its edges using a distance-based method;
3. for each Ω, compute its skeleton S_Ω and feature transform of the skeleton FT_S;
4. for each cluster C, attract its edges towards S_Ω using FT_S;
5. repeat from step 1 or 2 to reach the desired bundling level.

We start with an unbundled graph $G = (V,E)$ with nodes V and edges E. Node positions $v_i \in \mathbf{R}^2$ come either from input data or from laying out G with any existing method *e.g.* spring embedders [16]. Edges $e_i \in E$ are sampled as a set of points connected by lines. The start point e_i^s and end point e_i^s of an edge are the positions of the nodes the edge connects. Edge points may come from input data, *e.g.* when bundling a graph with explicit edge geometry, or by uniformly sampling the line segments (e_i^s, e_i^e). Our algorithm iteratively updates these edge points to bundle edges, as explained next.

3.1 Clustering

To obtain elongated 2D shapes, needed for our bundling described next in Sec. 3.3, we first cluster edges using a similarity metric which groups same-direction, spatially close, edges, using the clustering method described in [27]. We have tested several clustering algorithms: hierarchical bottom-up agglomerative (HBA) clustering using full, centroid, single, and average linkage, and k-means clustering, both with Euclidean and statistical correlation (Pearson, Spearmans rank, Kendalls τ) distances. HBA with full linkage and Euclidean distance given by

$$d(e_i, e_j) = \sqrt{\sum_{k=1}^{N} \|e_{ik} - e_{jk}\|^2} \tag{1}$$

where $e_{ik,k\in\overline{1,N},N\in[50,100]}$ are uniformly spaced along the edges gives clusters with geometrically close edges which naturally follow the graph structure. d can be easily adapted to use edge data attributes [27]. Using the same N for all edges removes edge length bias. HBA delivers a dendrogram $D = \{C_i\}$ with the edge set E as leaves and similarity (linkage) values $d(C)$, equal to the full linkage of cluster C increasing from root to leaves. We next do a partition $P = \{C_i \in D | d(C_i) < \delta\}$ of E based on a similarity value δ, set as explained further in Secs. 3.5 and 4.

3.2 Shape construction

For any shape $\Phi \subset \mathbf{R}^2$, we define its distance transform $DT_\Phi : \mathbf{R}^2 \to \mathbf{R}_+$ as

$$DT_\Phi(\mathbf{x} \in \mathbf{R}^2) = \min_{\mathbf{y}\in\Phi} \|\mathbf{x} - \mathbf{y}\| \tag{2}$$

For a cluster $C = \{e_i\}$, denote the set of polylines corresponding to its edges e_i by $\Delta(C) \subset \mathbf{R}^2$. Given a distance value ω, we construct a compact 2D shape $\Omega \subset \mathbf{R}^2$ surrounding $\Delta(C)$ as

$$\Omega = \{\mathbf{x} \in \mathbf{R}^2 | DT_{\Delta(C)}(\mathbf{x}) \leq \omega\} \tag{3}$$

where $DT_{\Delta(C)}$ is the distance transform of the drawing $\Delta(C)$ of C. The shape's boundary $\partial\Omega$ is the level set of value ω of $DT_{\Delta(C)}$. This is equivalent to dilating $\Delta(C)$ with a distance ω set to a small fraction (*e.g.* 0.05) of the bounding box of G.

3.3 Skeleton construction

Given a shape Ω computed from an edge cluster, we next compute its skeleton S_Ω defined as

$$S_\Omega = \{\mathbf{x} \in \Omega | \exists \mathbf{y}, \mathbf{z} \in \partial\Omega, \mathbf{y} \neq \mathbf{z}, \|\mathbf{x} - \mathbf{y}\| = \|\mathbf{x} - \mathbf{z}\| = DT_{\partial\Omega}(\mathbf{x})\} \tag{4}$$

i.e. the set of points in Ω which admit at least two different and closest points on $\partial\Omega$, also called feature points. Given S, we now compute its so-called one-point feature transform $FT_S : \mathbf{R}^2 \to \mathbf{R}^2$, defined as

$$FT_S(\mathbf{x}) = \{\mathbf{y} \in S | DT_S(\mathbf{x}) = \|\mathbf{x} - \mathbf{y}\|\} \tag{5}$$

i.e. one of the feature points of $\mathbf{x}$. The skeleton is the identity set of FT_S, *i.e.* $\forall \mathbf{x} \in S, FT_S(\mathbf{x}) = \mathbf{x}$. Note that Eqn. 5 uses the distance transform DT_S of the skeleton S, and not the distance transform $DT_{\partial\Omega}$ of the shape. We compute distance transforms, one-point feature transforms, and skeletons in discrete image (screen) space. This allows efficient implementation (Sec. 4) and also using the skeleton for edge bundling, as described next.

3.4 Edge attraction

We start with some observations. First, a cluster $C = \{e_i\}$ contains only spatially close edges. By construction, the skeleton S of a cluster is locally centered with respect to the (similar) edges in that cluster, *i.e.* a good candidate to bundle towards. Secondly, $FT_S(\mathbf{x}) - \mathbf{x}$ gives, for each point $\mathbf{x} \in \mathbf{R}^2$, the direction vector from $\mathbf{x}$ to the closest skeleton point to $\mathbf{x}$, *i.e.* the direction to bundle towards. We use these observations to bundle the edges $e_i \in C$ by attracting them towards S as follows.

First, we compute all branch termination points, or *tips*, $T = \{\mathbf{t}_i\}$ of S using a simple and efficient 3×3 pixel template-based method [17]. Next, we compute all skeleton paths $\Pi = \{\pi_i \subset S\}$ between any two tips $\mathbf{t}_i$ and $\mathbf{t}_j$. Paths are stored as pixel chains and are found using depth-first search from each $\mathbf{t}_i$ on the skeleton pixel-adjacency-graph. For each $e_i \in C$ with start and end points e_i^s and e_i^e respectively, we select a skeleton path $\pi(e_i) \in \Pi$ so that $\{FT_S(e_i^s), FT_S(e_i^e)\} \subset \pi(e_i)$, *i.e.* a path passing through the feature points of both edge end points. If there are several such paths in Π, we pick any one of them, the particular choice having no further impact.

We now use $\pi(e_i)$ to bundle e_i along the skeleton, as follows. Consider a point $\mathbf{x} \in e_i$ located at arc-length distance $\lambda(\mathbf{x})$ from $\mathbf{e}_i^s$. We move $\mathbf{x}$ towards $FT_S(\mathbf{x})$ with a distance which is large if $\mathbf{x}$ is far away from $FT_S(\mathbf{x})$ or close to the middle of the edge:

$$\mathbf{x}^{new} = \left[1 - \alpha\phi\left(\frac{\lambda(\mathbf{x})}{\lambda(\mathbf{e}_i^e)}\right)\right]\mathbf{x} + \alpha\phi\left(\frac{\lambda(\mathbf{x})}{\lambda(\mathbf{e}_i^e)}\right) FT_S(\mathbf{x}) \tag{6}$$

$\alpha \in [0,1]$ controls bundling tightness. The function $\phi : [0,1] \to [0,1]$ defined as

$$\phi(t) = [2\ \min(t, 1-t)]^K \tag{7}$$

modulates the motion amount: The edge's end points $\mathbf{e}_i^s$ and $\mathbf{e}_i^e$ do not move at all; points close to these end points move less; and points around the middle of the edge move most. This produces the curved edge profile we require for bundling, and also keeps edge end points fixed. The parameter K controls how much edges twist when bundled. Values of $K \in [3,6]$ give very similar results to known bundling methods *e.g.* [14, 15, 20]. For any $\mathbf{x} \in S$, $FT_S(\mathbf{x}) = \mathbf{x}$ (Sec. 3.3), so for such points we have

$\mathbf{x}^{new} = \mathbf{x}$ (Eqn. 6), *i.e.* points which have reached the skeleton, the extreme bundling location, stop moving.

Equation 6 is equivalent to advecting edge points $\mathbf{x}$ in the gradient field $-\nabla DT_S$. Distance transforms of any shape except a straight line have div $\nabla DT_S \neq 0$ [23]. Hence, our attraction typically shortens and/or lengthens edges. Since we compute the edge points $\mathbf{x}$ in Eqn. 6 by uniformly sampling edges in arc-length space, attraction removes points where edges contract (div $\nabla DT_S < 0$) and inserts points where edges dilate (div $\nabla DT_S > 0$) as needed.

3.5 Iterative edge bundling

Applying the clustering, shape construction, and edge attraction steps outlined above yields a small amount of bundling of a graph layout. We repeat this process iteratively until a user-specified number of iterations I is reached. More iterations yield tighter bundles. This process is strictly monotonic, *i.e.* edges can only get closer to their clusters' skeletons (hence to each other) by construction, as explained below (see also Fig. 1).

For the first clustering, we use a high similarity threshold δ to guarantee elongated, thin, clusters for any edge spatial distribution in the input graph (Sec. 3.1). This is essential for getting the initial bundling under way. Indeed, weakly coherent clusters would contain edges that intersect each other at large angles; hence the shapes surrounding them, and their skeletons, would be meaningless bundling cues. For subsequent iterations, we decrease δ and recluster each few $(3..5)$ iterations. This produces fewer, increasingly larger, clusters. However, these clusters are *locally* elongated, since they contain already partially bundled edges. Hence, coarsening the clustering will not group unrelated edges. The overall effect is bottom-up bundling: First, the closest edges get bundled into fine-scale local bundles, which next increasingly merge into coarser-scale bundles.

Similarly, we decrease α during the iterative process. Initial large α values yield strongly coherent initial bundles, needed for clustering stability as outlined above. Subsequent relaxed α values allow edges in more complex, larger, bundles to adjust themselves. Concrete values for δ and α are given in Sec. 4.2.

4 Implementation

4.1 Image-based operations

We compute shapes, skeletons, skeleton tips, and distance and feature transforms in an image-based setting. For this, we use a Nvidia CUDA 1.1 implementation of exact Euclidean distance-and-feature transforms [4]. We extended this technique to compute robust skeletons based on the augmented fast marching method (AFMM) in [28]. The AFMM *guarantees* that no spurious branches appear due to boundary perturbations, which in turn guarantees stable bundling cues. Table 1 shows statistics for several graphs bundled with our method. Table 2 shows timings per algorithm phase.

Our CUDA implementation takes 4 milliseconds per distance, feature transform, and skeletonization for an image of 800 by 800 pixels on an Nvidia GT 330M GT card, in line with figures reported in [4]. A CPU-only implementation such as [28] is roughly 100 times slower. Clustering is also fast. The CPU implementation in [8] constructs the complete dendrogram of a graph of 10K edges in 0.1 seconds on a 2.8 GHz PC. We also tried the GPU-based clustering in [5], which is roughly 10 to 15 times faster. Note that only a few clustering passes are needed for a complete layout (Sec. 3.5). All in all, our method takes 5 to 30 seconds for producing a final layout for the graphs we tested (Tab. 1, right column), *i.e.* 25 milliseconds per cluster times the total number of clusters processed during the $I = 10$ iterations plus the clustering time.

Graph	Nodes	Edges	Clusters/iteration			Time
			$I = 1$	$I = 5$	$I = 10$	(sec.)
Airlines	235	2099	90	15	9	6.3
Migrations	1715	9780	57	14	7	4.1
Radial	1024	4021	94	30	24	7.4
France air	34550	17275	207	40	26	29.2
Poker	859	2127	86	28	23	5.2

Table 1: Graph statistics for datasets used in this paper.

Graph ($I = 5$)	Tips	Points	Inflation (ms)	Holes (ms)	Skel. (ms)	Paths (ms.)	Attraction (ms)
Airlines	22	8388	77	120	314	98	20
Migrations	28	9780	78	134	339	170	77
Radial	14	21580	80	96	357	45	17
France air	34	23759	81	148	374	222	88
Poker	28	2385	64	117	238	146	13
CUDA			2	8	2	< 12	3

Table 2: Bundling performance. Per-cluster average time for the different phases of our method. First rows show CPU timings. Last row shows CUDA timings (uniform for the tested graphs).

4.2 Parameter setting

Our method has a few parameters: clustering similarity threshold δ, edge advection factor α, and number of iterations I.

Clustering similarity threshold δ: Specifies the level at which we cut the cluster dendrogram to obtain edge sets to bundle at the current iteration (Sec. 3.1). We set δ as a linearly decreasing function on the iteration number $t \in [1, I]$ from $\delta(1) = 0.95$ to $\delta(I) = 0.7$. This yields strongly coherent clusters in the first iteration, regardless of initial edge positions, and also *locally* strongly coherent clusters in the subsequent iterations (Sec. 3.5).

Edge advection factor α: Values $\alpha \in (0, 1)$ control how much edges approach the skeleton at one iteration. Too high values yield tight bundles and convergence in few iterations, which is fine for graphs having relatively grouped edges, but cannot declutter complex graphs. Too low values allow the process to adapt itself better to newly discovered clusters as the edges approach each other, but convergence requires more iterations. In

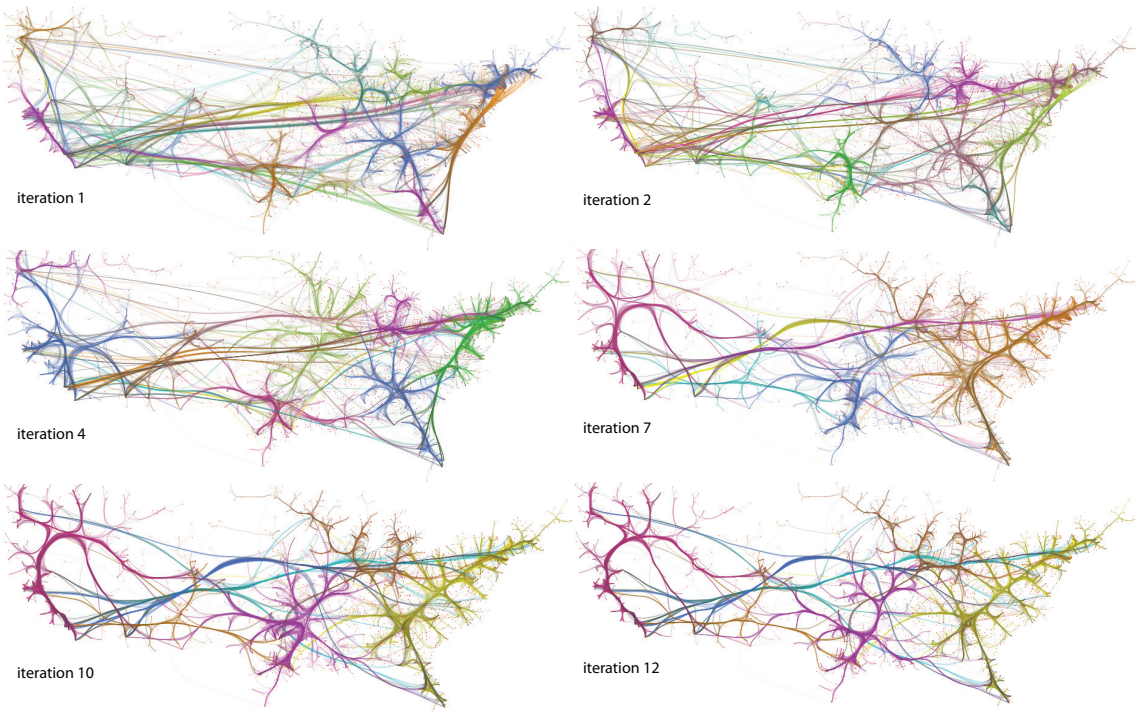

Figure 1: Iterative bundling of the US migrations graph. Colors indicate edge clusters (see Sec. 3.5).

practice, we set α as a linearly decreasing function of the iteration number from $\alpha(0) = 0.9$ to $\alpha(I) = 0.2$.

Number of iterations: We obtained tight bundles of a few pixels wide after $I \in [10, 15]$ iterations for all studied graphs. This is expectable since $(1-\alpha)^I$ becomes very small for $\alpha < 1, I > 10$. In practice, we always set $I = 10$.

The method is not sensitive to precise parameter settings. We explain this by the stability of the inflated shape skeletons to small local variations of the positions of edges, and the smoothing effect of the entire iterative process on the layout.

5 Applications

Figure 2 compares the SBEB with existing bundling methods for several graphs[1] (see also Tab. 1). Images (a,b) show an air traffic graph (nodes are city locations, edges are flight paths). Images (c,d) show a graph of poker players from a social network. Edges encode people that played against each other. The node layout is done with a spring embedder [3]. Given the average node degree and node layout used, related nodes tend to form similar-size cliques. Bundling further simplifies this structure; bundles encode sets of players which played against each other. Images (e-h) show the known US migrations graph bundled with the WR, GBEB, FDEB, and our method (SBEB). Overall, SBEB produces strong bundling, and emphasizes the structure of connections between groups of close cities (due to the skeleton layout cues). Less bundling can be achieved by less iterations (Fig. 1). Adjusting the bundling parameters (Sec. 4.2), SBEB can create bundling styles similar to either GBEB (higher bundle curvatures, more emphasis on the graph structure) or FDEB (smoother bundles). Images (i,j) show the US airlines graph bundled with the FDEB and SBEB respectively. SBEB generates stronger bundling (more overdraw) but arguably less clutter. Note also that SBEB generates tree-like bundle structures which is useful when the exploration task at hand has an inherent (local) hierarchical nature, *e.g.* see how traffic connections merge into and/or split from main traffic routes.

[1] More examples are available at www.cs.rug.nl/Shapes/SBEB

6 Discussion

Our method (SBEB) compares to related techniques as follows.

Generality: SBEB handles directed or undirected graphs. By default, we assume directed graphs. Edges between the same sets of nodes in opposite directions will belong to different clusters, hence create different bundles. To treat undirected graphs, we symmetrize the edge similarity function (Eqn. 1).

Structured look: Except HEB, other bundling methods do not emphasize a structured, tree-like, look of the bundles, since there is no explicit bundle hierarchy. SBEB models hierarchy by the cluster skeletons (at fine level) and by the simplified cluster structures (at coarse level).

Robustness: SBEB operates robustly on all graphs we tried it on, *i.e.* yields a set of stable skeletons and bundling converges towards a stable state. This is explained by the inherent robustness of the skeletonization method used (Sec. 3.3). Intuitively, adding or removing a small number of nodes or edges will not change the bundling since the distance-based shapes and their skeletons are robust to small changes.

Speed and simplicity: Since image-based and using a GPU implementation, SBEB is considerably faster than [15] and slightly

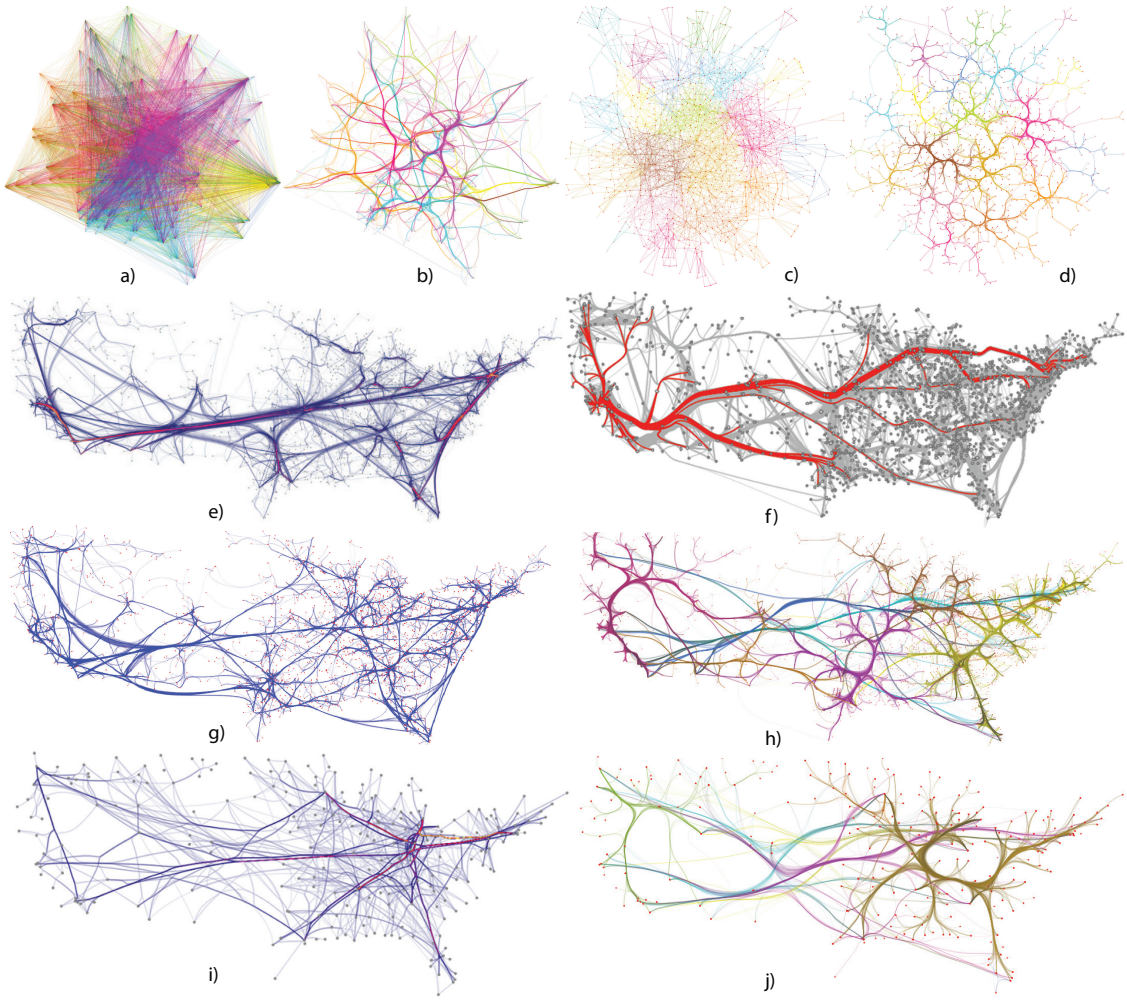

Figure 2: Air traffic graph (a: original, b: bundled). Poker graph (c: original, d: bundled). Migrations graph (e: FDEB, f: GBEB, g: WR, h: SBEB). Airlines graph (i: FDEB, j: SBEB). Colors in (a-d,h,j) indicate clusters (displayed for method illustration only).

faster than [20]. We note that it is not clear if the timings reported in [20] include also the cost of computing the Voronoi diagram underlying the grid graph. The only faster bundling method we know is MINGLE [12], which takes 1 second for the US migrations graph and 0.1 seconds for the US airlines graph, in contrast to our 4.1 seconds and 6.3 seconds respectively. MINGLE and SBEB share some resemblance in bottom-up aggregation of edges, but also have some differences. MINGLE compares edges essentially based on endpoint positions, whereas we use the entire edge trajectory, which should allow us to bundle graphs with curved edges better. The complexity of MINGLE is $O(|E|log|E|)$ for a graph with E edges, whereas SBEB is $O(|C|)$ where C is the average cluster size. A better cluster selection than our current iso-linkage cut in the cluster tree (Sec. 3.1) would reduce $|C|$ and thus make SBEB faster. SBEB works entirely image-based rather than by a combination of hierarchical mesh-based and image-based data structures. Our CUDA skeleton code is available at [26].

Algorithmics: In contrast to FDEB [15] which bundles edge pairs iteratively, we bundle increasingly larger edge clusters along their skeletons in one single step. In the limit, SBEB behaves like FDEB, *i.e.* if we treat only the most cohesive leaf cluster at each iteration. This is impractical, as it would artificially increase computational costs without any benefits. While Lambert *et al.* [20] use shortest paths in a node-based grid graph to route edges, SBEB uses only edge information. Distance fields and skeletons are also used in [27] for *shading* cues, whereas we use skeletons to actually compute edge *layouts*. In comparison to [21], where bundles split in exactly two sub-bundles, our bundle splits can have any degree, as implied by the underlying skeletons.

Limitations: There is no hard reason why SBEB should be preferable to other bundling heuristics, apart from the intuition that a skeleton represents the local center of a shape. The quality of our layouts (or any other bundled layout) is still to be judged subjectively. Any bundling inherently looses information: edges are overdrawn, so cannot be identified separately; and edge directions are distorted. Hence, bundling should be used for those applications where one is interested in coarse-scale connectivity patterns and/or when explicit graph simplification *e.g.* is not an option. SBEB can be adapted to incorporate additional bundling constraints *e.g.* maximal deformation

of certain edges: the skeletons provide only bundling *cues* but the attraction phase can decide whether, and how much, to bundle any given edge. In the long term, it is interesting to use shape perception results from computer vision [6] to quantitatively reason about the quality of a bundled layout. Our image-based approach may prove more amenable to quantitative analysis than other bundling heuristics which are harder to describe in terms of imaging operators. However, this is a challenging task and requires further in-depth study.

7 Conclusion

We have presented a new method for creating bundled layouts of general graphs. Using the centeredness property of 2D skeletons, we create elongated shapes from a graph with given node positions, and use skeletons as guidelines to bundle similar edges. Our layout amounts to a sequence of edge clustering and image processing operations which can be efficiently implemented on the GPU to achieve higher performance than existing comparable methods.

We plan to exploit skeleton properties to generate bundling variations. Modifying the Euclidean distance metric would yield layouts similar to cartographic diagrams [25]. We plan to use bundle-bundle and bundle-node distance fields to globally optimize the layout for maximal readability and incorporate spatial constraints like labels, bundle crossing minimization, and node-edge overlap reduction. In the long run, we plan to study the optimality criteria of bundled layouts by using existing results from shape perception in computer vision which are directly applicable to our skeleton-based layout method.

References

[1] J. Abello, F. van Ham, and N. Krishnan. AskGraphView: A large graph visualisation system. *IEEE TVCG*, 12(5):669–676, 2006.

[2] D. Archambault, T. Munzner, and D. Auber. Grouse: Feature-based and steerable graph hierarchy exploration. In *Proc. EuroVis*, pages 67–74, 2007.

[3] D. Auber. Tulip visualization framework, 2011. `tulip.labri.fr`.

[4] T. Cao, K. Tang, A. Mohamed, and T. Tan. Parallel banding algorithm to compute exact distance transform with the GPU. In *Proc. ACM SIGGRAPH Symp. on Interactive 3D Graphics and Games*, pages 134–141, 2010.

[5] D. Chang, M. Kantardzic, and M. Ouyang. Hierarchical clustering with cuda/gpu. In *Proc. ISCA*, pages 130–135, 2009.

[6] L. Costa and R. Cesar. *Shape analysis and classification: Theory and practice*. CRC Press, 2000.

[7] W. Cui, H. Zhou, H. Qu, P. Wong, and X. Li. Geometry-based edge clustering for graph visualization. *IEEE TVCG*, 14(6):1277–1284, 2008.

[8] M. de Hoon, S. Imoto, J. Nolan, and S. Myiano. Open source clustering software. *Bioinformatics*, 20(9):1453–1454, 2004.

[9] M. Dickerson, D. Eppstein, M. Goodrich, and J. Meng. Confluent drawings: Visualizing non-planar diagrams in a planar way. In *Proc. Graph Drawing*, pages 1–12, 2003.

[10] T. Dwyer, K. Marriott, and M. Wybrow. Integrating edge routing into forcedirected layout. In *Proc. Graph Drawing*, pages 8–19, 2007.

[11] G. Ellis and A. Dix. A taxonomy of clutter reduction for information visualisation. *IEEE TVCG*, 13(6):1216–1223, 2007.

[12] E. Gansner, Y. Hu, S. North, and C. Scheidegger. Multilevel agglomerative edge bundling for visualizing large graphs. In *Proc. PacificVis*, pages 187–194, 2010.

[13] E. Gansner and Y. Koren. Improved circular layouts. In *Proc. Graph Drawing*, pages 386–398, 2006.

[14] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE TVCG*, 12(5):741–748, 2006.

[15] D. Holten and J. J. van Wijk. Force-directed edge bundling for graph visualization. *Comp. Graph. Forum*, 28(3):670–677, 2009.

[16] I.Tollis, G. D. Battista, P. Eades, and R. Tamassia. *Graph drawing: Algorithms for the visualization of graphs*. Prentice Hall, 1999.

[17] R. Klette and A. Rosenfeld. *Digital geometry: Geometric methods for digital picture analysis*. Morgan Kaufmann, 2004.

[18] I. Kovacs, A. Feher, and B. Julesz. Medial-point description of shape: A representation for action coding and its phychophysical correlates. *Vision research*, 38:2323–2333, 1998.

[19] A. Lambert, R. Bourqui, and D. Auber. 3D edge bundling for geographical data visualization. In *Proc. Information Visualisation*, pages 329–335, 2010.

[20] A. Lambert, R. Bourqui, and D. Auber. Winding roads: Routing edges into bundles. *Comp. Graph. Forum*, 29(3):432–439, 2010.

[21] D. Phan, L. Xiao, R. Yeh, P. Hanrahan, and T. Winograd. Flow map layout. In *Proc. InfoVis*, pages 219–224, 2005.

[22] H. Qu, H. Zhou, and Y. Wu. Controllable and progressive edge clustering for large networks. In *Proc. Graph Drawing*, pages 399–404, 2006.

[23] K. Siddiqi, S. Bouix, A. Tannenbaum, and S. Zucker. Hamilton-Jacobi skeletons. *IJCV*, 48(3):215–231, 2002.

[24] K. Siddiqi and S. Pizer. *Medial Representations: Mathematics, Algorithms and Applications*. Springer, 1999.

[25] R. Strzodka and A. Telea. Generalized distance transforms and skeletons in graphics hardware. In *Proc. VisSym*, pages 221–230, 2004.

[26] A. Telea. CUDA skeletonization and image processing toolkit, 2011. `www.cs.rug.nl/~alext/CUDASKEL`.

[27] A. Telea and O. Ersoy. Image-based edge bundles: Simplified visualization of large graphs. *Comp. Graph. Forum*, 29(3):543–551, 2010.

[28] A. Telea and J. J. van Wijk. An augmented fast marching method for computing skeletons and centerlines. In *Proc. VisSym*, pages 251–259, 2002.

[29] F. vam Ham. Using multilevel call matrices in large software projects. In *Proc. InfoVis*, pages 227–232, 2003.

[30] R. van Liere and W. de Leeuw. GraphSplatting: Visualizing graphs as continuous fields. *IEEE TVCG*, 9(2):206–212, 2003.

[31] H. Zhou, X. Yuan, W. Cui, H. Qu, and B. Chen. Energy-based hierarchical edge clustering of graphs. In *Proc. PacificVis*, pages 55–62, 2008.